UNIVERSITY OF TEXAS AT DALLAS

MIS 6308 0W1 – SYSTEM ANALYSIS AND PROJECT MANAGEMENT

Professor – Srinivasan Raghunathan

Project Report: SmartGro IOT-Based Restaurant Inventory Management System

Presented By (Group 6):

- Aditya Bhoj
- Radhika Dube
- Sai Bhavana Marada

Table of Contents

Problem Statement

Problem

Restaurant Inventory Waste
Restaurants often face significant challenges in managing their inventory of groceries with varying expiry dates. This inefficient management leads to food waste when items expire before they can be used. The process of tracking expiration dates and planning menus accordingly is often manual and prone to errors. This not only results in financial losses but also contributes to environmental issues through increased food waste. There's a need for a smart system that can help restaurants better manage their inventory, reduce waste, and optimize their menu planning process.

Inefficient Menu Planning
Chefs frequently struggle to adjust their menus based on the current inventory status, especially regarding items approaching their expiry dates. Without real-time visibility into inventory expiration dates, chefs may miss opportunities to use ingredients before they spoil. This leads to unnecessary waste and limits the chef's ability to create specials or adjust menus to make the best use of available ingredients. A system that provides timely information about inventory status could significantly improve menu planning and reduce waste.

Lack of Real-Time Inventory Insights
Restaurant managers often lack up-to-date information about their inventory status. Traditional inventory management systems may not provide real-time data, leading to overstocking or understocking of items. This can result in either excessive waste or shortages that affect menu offerings. Real-time inventory tracking would allow for more precise ordering and better overall inventory management.

Objectives

IoT-Enabled Inventory Monitoring (IIM)
SmartGro will implement an IoT-based system using smart sensors to continuously monitor the status of grocery items in the restaurant's inventory. These sensors will track expiration dates and storage conditions, providing real-time data to the inventory management system. By leveraging this technology, restaurants can maintain an accurate, up-to-date view of their inventory status always.

Smart Expiry Alert and Menu Suggestion System (SEAMSS)
The system will include a notification feature that alerts chefs about items approaching their expiry dates. This will be coupled with a menu suggestion algorithm that proposes dishes or specials incorporating these near-expiry items. By providing timely alerts and creative menu ideas, SEAMSS aims to minimize food waste and inspire chefs to make the most of available ingredients.

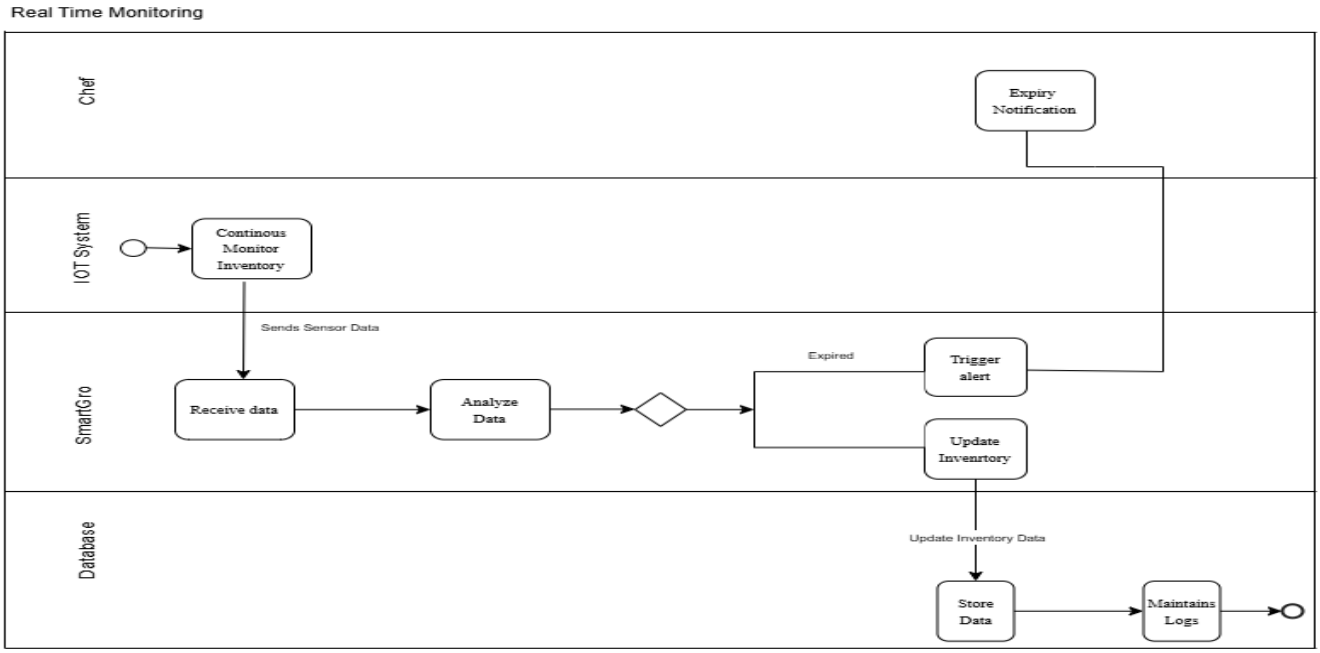
Dynamic Inventory Analytics (DIA)
SmartGro will develop an analytics tool that processes data from the IoT sensors to provide insights on inventory usage patterns, waste trends, and ordering recommendations. This feature will help restaurant managers optimize ordering processes, reduce overstocking, and make data-driven menu offerings and inventory management decisions.

Scope

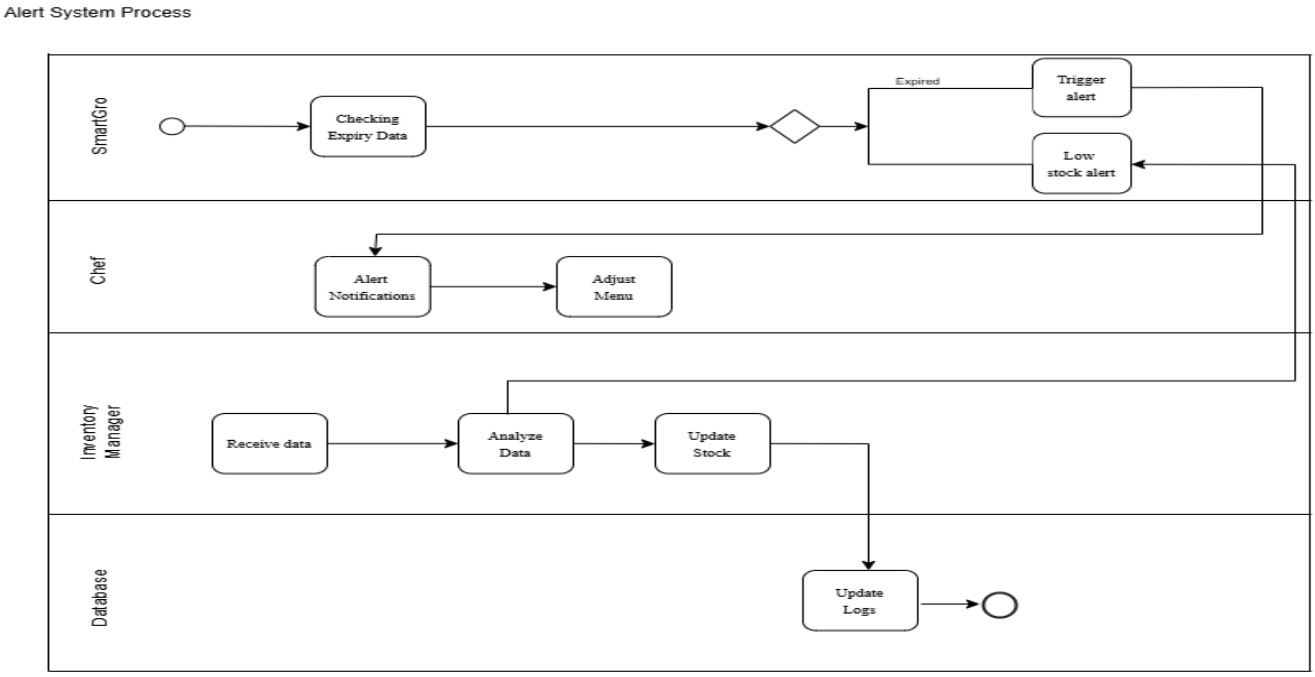
- The estimated cost for the development of the SmartGro IoT-Based Inventory Management System is approximately $120,000.
- The project is expected to be completed within 6 months.
- The development team will consist of IoT specialists, software developers, data analysts, UI/UX designers, and project managers to ensure an efficient and seamless development process.
- Additional database storage will be required to store inventory details, IoT sensor readings, alerts, menu suggestions, and maintenance logs.
- The system will integrate with existing restaurant management platforms and supplier databases to streamline inventory tracking, order management, and alert notifications.

Business Process Model Notation (BPMN)

1. Real-Time Monitoring

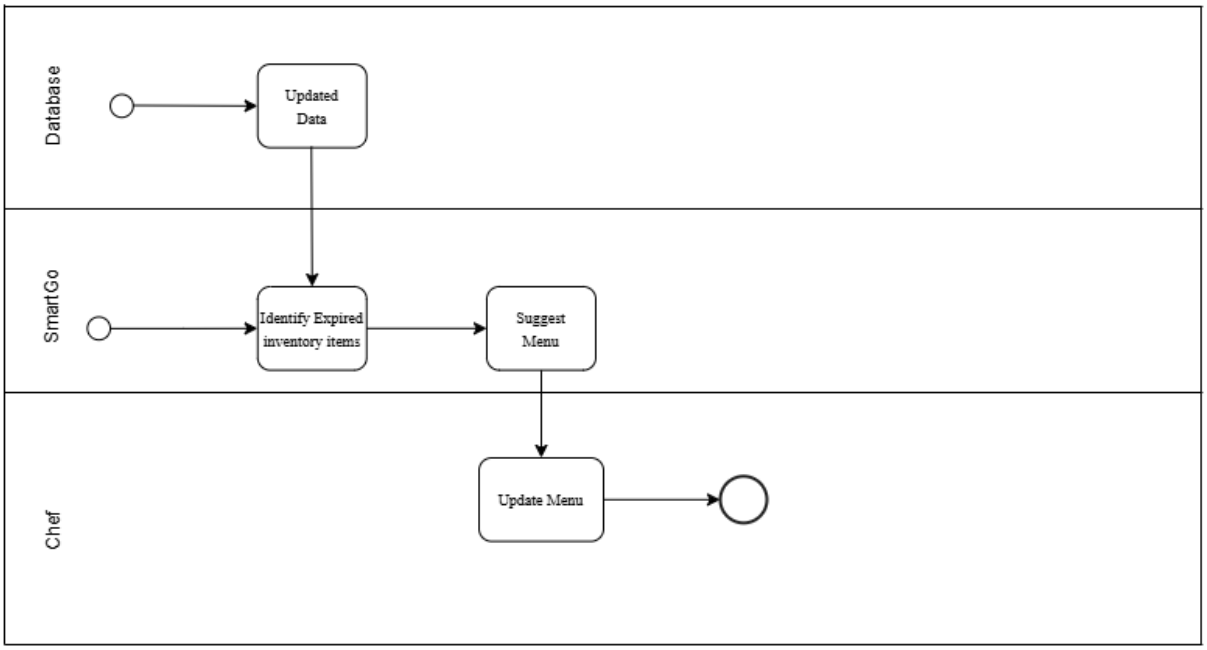


2. Alert System Process

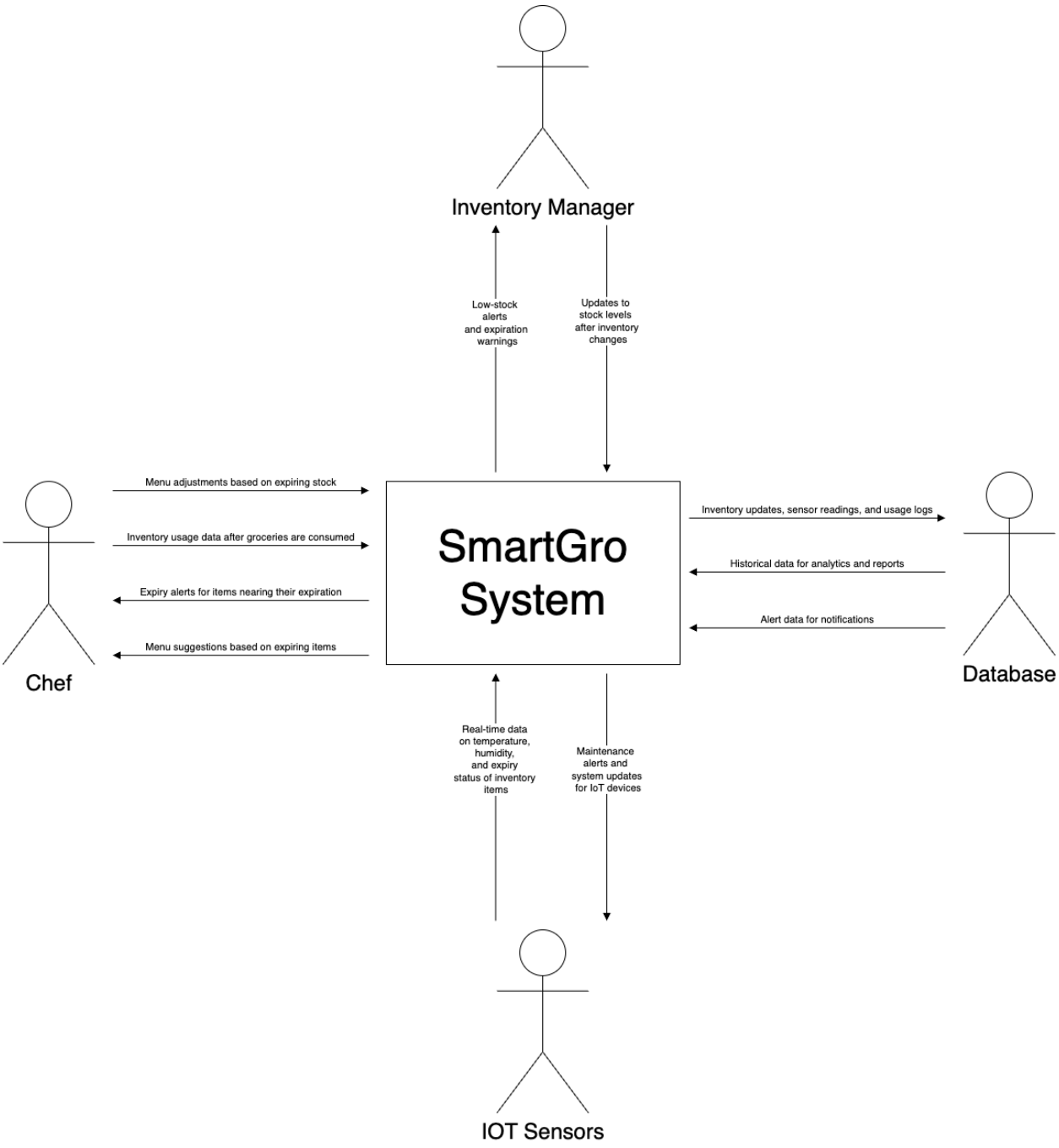


3. Menu Adjustment Process

Menu Adjustment Process

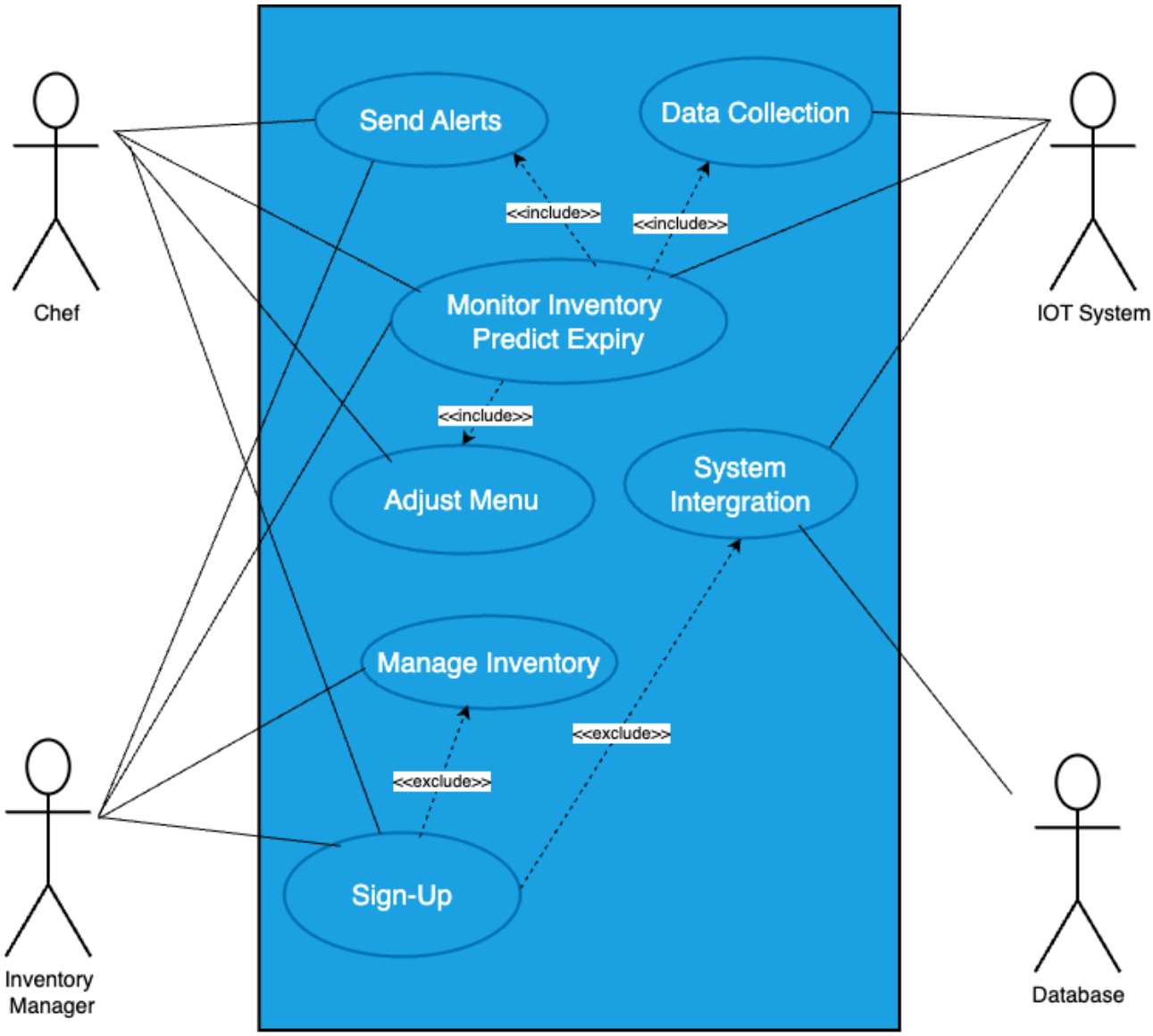


Context Diagram



Process Model

Use-Case Diagram



Use Case Descriptions

Use-Case 1: **Sign Up**

Use Case Name: Sign Up
Primary Actor: Chef, Inventory Manager
Stakeholders: • Chef: Registers on the SmartGro platform to receive real-time inventory alerts and menu suggestions. • Inventory Manager: Signs up to manage inventory, track usage, and optimize stock levels. • SmartGro System: Maintains user accounts and permissions for secure access.
Brief Description: The SmartGro system enables chefs and inventory managers to create user accounts on its platform for accessing the IoT-enabled inventory management features.
Trigger: The user selects the option to create a new account.
Normal Flow of Events: 1. The user navigates to the sign-up page. 2. The user enters relevant information (name, role, email, password, contact information). 3. The system validates the entered data for completeness and correctness (e.g., email format, password strength). 4. If data is valid, the system creates a new user account and stores it in the database. 5. The system confirms successful registration by sending a welcome email or displaying a confirmation message.
Exception Flow: 1. Condition: If the user already has an account, the system prompts the user to log in instead.

Use-Case 2: **Monitor Inventory and Predict Expiry**

Use Case Name: Monitor Inventory and Predict Expiry
Primary Actor: IoT System
Stakeholders: • **Chef:** Needs to know which items are nearing expiry to avoid wastage. • **Inventory Manager:** Uses the system's alerts to manage stock and make decisions about replenishment. • **Database:** Stores inventory data and expiry status.
Brief Description: The SmartGro system continuously monitors inventory through IoT sensors, tracks expiry dates, and predicts which items will expire soon.
Trigger: Inventory data is sent from IoT sensors to the system.
Normal Flow of Events: 1. IoT sensors capture data about stored groceries. 2. Data is sent to the SmartGro system in real time. 3. SmartGro checks the expiry dates of items. 4. If items are nearing expiry, the system predicts the expiration and triggers alerts. 5. Alerts are sent to the chef to notify them of items to be prioritized. 6. The database is updated with the latest inventory status and usage logs.
Exception Flow: 1. If IoT sensors fail to transmit data, the system will request manual input from the inventory manager. 2. If no items are near expiry, the system takes no further action and only updates the inventory log.

Use-Case 3: **Adjust Menu Based on Expiry Alerts**

Use Case Name: Adjust Menu Based on Expiry Alerts
Primary Actor: Chef
Stakeholders: • **Chef:** Adjusts the menu to use items nearing expiry. • **Inventory Manager:** Uses the updated menu to manage stock.
Brief Description: The system sends expiry alerts to the chef, suggesting which items to prioritize. The chef adjusts the menu accordingly.
Trigger: The system identifies items nearing expiry.
Normal Flow of Events: 1. The SmartGro system sends expiry alerts with suggestions to the chef. 2. The chef reviews the items that are close to expiry. 3. Based on the suggestions, the chef updates the menu. 4. Inventory usage data is returned to the system after the groceries are consumed.
Exception Flow: 1. If the chef cannot use the items, the system alerts the inventory manager to remove expired stock.

Use-Case 4: **Manage Inventory**

Use Case Name: Manage Inventory
Primary Actor: Inventory Manager
Stakeholders: • **Chef:** Relies on updated inventory data to plan menus. • **Database:** Stores the latest inventory records.
Brief Description: The inventory manager updates stock levels and tracks usage data to maintain accurate inventory.
Trigger: Stock levels need to be updated after items are consumed.
Normal Flow of Events: 1. The inventory manager logs into the SmartGro system. 2. The manager updates stock levels based on usage data. 3. If required, the manager places orders for new stock. 4. The database is updated with the new inventory status. 5. The system generates alerts if stock levels fall below a threshold.
Exception Flow: 1. The system notifies the inventory manager if an error occurs during the update.

Use-Case 5: **Send Alerts and Recommendations**

Use Case Name: Send Alerts and Recommendations
Primary Actor: SmartGro System
Stakeholders: • **Chef:** Receives alerts to reduce wastage. • **Inventory Manager:** Uses recommendations to optimize stock.
Brief Description: The system sends alerts to the chef and recommendations to the inventory manager to reduce food waste.
Trigger: The system detects items nearing expiry or low stock levels.
Normal Flow of Events: 1. SmartGro checks the database for items nearing expiry or low stock. 2. Alerts are sent to the chef about items to prioritize. 3. Recommendations are sent to the inventory manager for replenishment. 4. The system updates the database with alert logs and recommendations.
Exception Flow: 1. If no action is needed, the system logs the data and sends no alert.

Use Case 6: **Collect Sensor Data**

Use Case Name: Collect Sensor Data
Primary Actor: IoT Sensors
Stakeholders: • **Chef:** Relies on sensor data for updated inventory conditions. • **Inventory Manager:** Uses sensor data to adjust stock. • **Database:** Stores real-time environmental data.
Brief Description: IoT sensors collect and transmit data on inventory conditions like temperature and humidity.
Trigger: Sensors detect environmental changes or reach reporting intervals.
Normal Flow of Events: 1. Sensors capture data on inventory conditions. 2. Data is transmitted to the database. 3. The system processes and logs sensor data. 4. Alerts are generated for any anomalies (e.g., temperature deviation).
Exception Flow: 1. If sensor data is not available, manual input is requested.

Use-Case 7: **Update Database**

Use Case Name: Update Database
Primary Actor: Database
Stakeholders: • **Chef:** Relies on updated inventory data. • **Inventory Manager:** Ensures accurate inventory records.
Brief Description: The system updates inventory and alert logs to maintain accurate records.
Trigger: Inventory data or alerts need updates.
Normal Flow of Events: 1. Inventory logs are updated after stock changes. 2. Alert logs are updated based on new alerts or resolutions. 3. User activity is logged for auditing.
Exception Flow: 1. Errors trigger a notification for manual corrections.

Data Dictionary

Use Case 1: Sign Up

- **User Data** = (Username + Firstname + Lastname + Role + Location + Phone_Number + Email_ID + Password)
- **User** = {UserID + User Data}
- **Role** = Chef | Inventory Manager

Use Case 2: Monitor Inventory and Predict Expiry

- **Inventory Data** = ItemID + Name + Category + Quantity + ExpiryDate + Temperature + Humidity + Location + StockLevel
- **Sensor Data** = SensorID + SensorType (Temperature/Humidity) + Location + Status + LastUpdated + Readings
- **Alert Data** = AlertID + ItemID + ExpiryDate + AlertDate + AlertMessage + AlertType (Expiry, LowStock)
- **Monitoring Data** = AlertID + ItemID + ExpiryDate + AlertDate + AlertMessage

Use Case 3: Adjust Menu Based on Expiry Alerts

- **Alert Data** = AlertID + ItemID + ExpiryDate + AlertDate + SuggestedMenuItems + AlertType + Message
- **Chef Data** = ChefID + Name + Contact + Email
- **Menu Data** = MenuID + SuggestedItems + LastUpdated
- **Usage Data** = ItemID + QuantityUsed + UsageDate + ChefID
- **Suggested Menu Items** = {ItemID + Name + Quantity + ExpiryDate}

Use Case 4: Manage Inventory

- **Inventory Manager Data** = ManagerID + Name + Contact + Department + Email
- **Inventory Data** = ItemID + Name + Category + Quantity + ExpiryDate + StockLevel + Location
- **Order Data** = OrderID + ItemID + OrderDate + Quantity + Status
- **Stock Report** = ReportID + DateGenerated + LowStockItems + ExpiringSoonItems + PreparedBy

Use Case 5: Send Alerts and Recommendations

- **Alert Data** = AlertID + ItemID + AlertType + AlertMessage + Status (Unread, Acknowledged) + Timestamp
- **Recommendations** = RecommendationID + ItemID + SuggestedActions (Order/Use) + PriorityLevel
- **Notification Data** = NotificationID + UserID + NotificationType (Email/SMS) + Status + Timestamp

Use Case 6: Collect Sensor Data

- **Sensor Data** = SensorID + Location + Status + Temperature + Humidity + LastUpdated
- **Environmental Data** = Temperature + Humidity + Timestamp
- **Monitoring Data** = {SensorID + ItemID + Temperature + Humidity + Status}

Use Case 7: Update Database

- **Database Data** = DatabaseID + ServerLocation + StorageSize + LastBackupDate
- **Inventory Log** = LogID + ItemID + Action + Timestamp + UpdatedBy
- **Alert Log** = AlertID + ItemID + AlertType + Status + Timestamp
- **User Data** = UserID + Role (Chef, InventoryManager) + ContactInfo

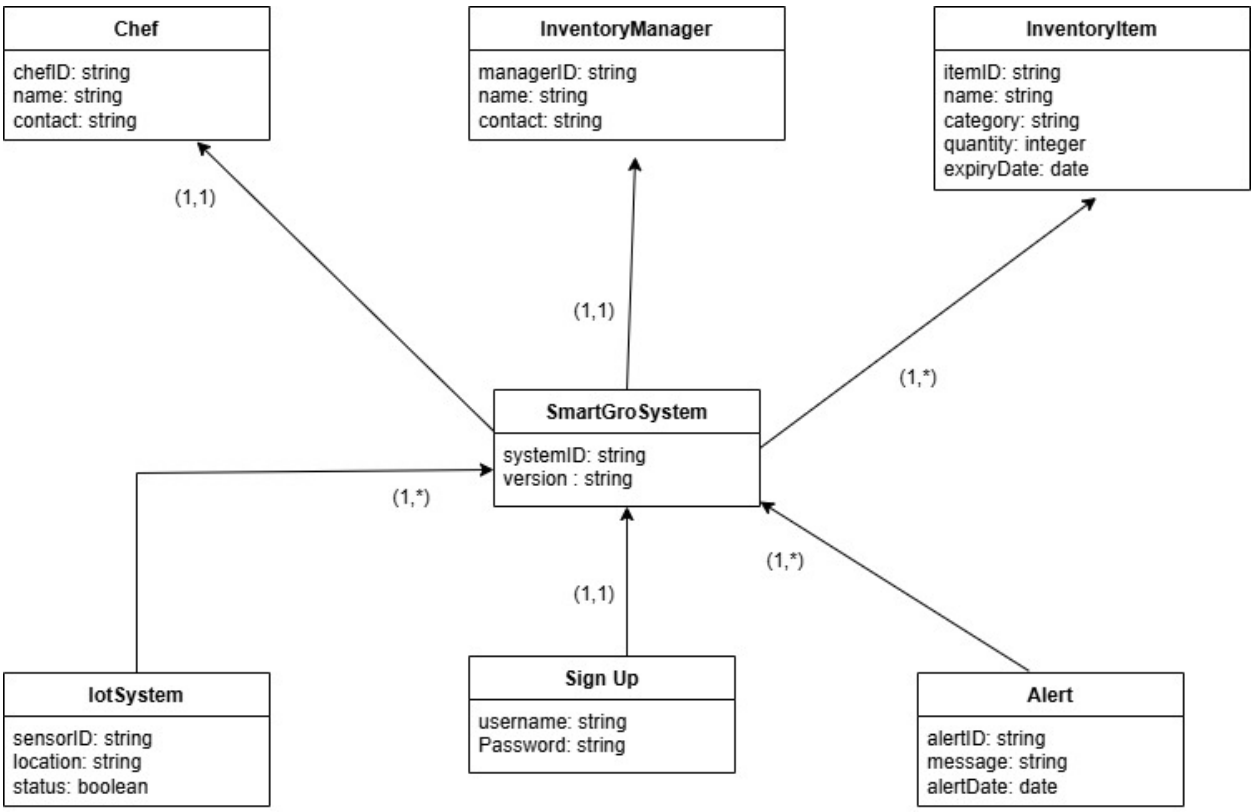
Inventory Item = data element
Item ID = data element
Item Name = data element
Category = data element
Quantity = data element
Unit of Measurement = data element
Expiry Date = data element
Storage Location = data element
Temperature = data element
Humidity = data element
Created At = data element
Sensor Data = data element

Sensor ID = data element
Sensor Type = data element
Sensor Location = data element
Sensor Status = data element
Reading ID = data element
Temperature Reading = data element
Humidity Reading = data element
Reading Timestamp = data element
User = data element
User ID = data element

User Name = data element
Role = data element
Email = data element
Password Hash = data element
Alerts = data element
Alert ID = data element
Alert Type = data element
Alert Message = data element
Alert Status = data element
Created At = data element
Orders = data element
Order ID = data element
Order Date = data element
User ID = data element
Order Status = data element
Total Cost = data element
Order Items = data element
Order Item ID = data element
Order ID = data element
Item ID = data element
Quantity Ordered = data element
Price at Order = data element
Usage Logs = data element
Log ID = data element
Item ID = data element
User ID = data element
Quantity Used = data element
Usage Date = data element
Maintenance Logs = data element
Maintenance ID = data element
Sensor ID = data element
Maintenance Date = data element
Maintenance Status = data element
Maintenance Details = data element

Data Model

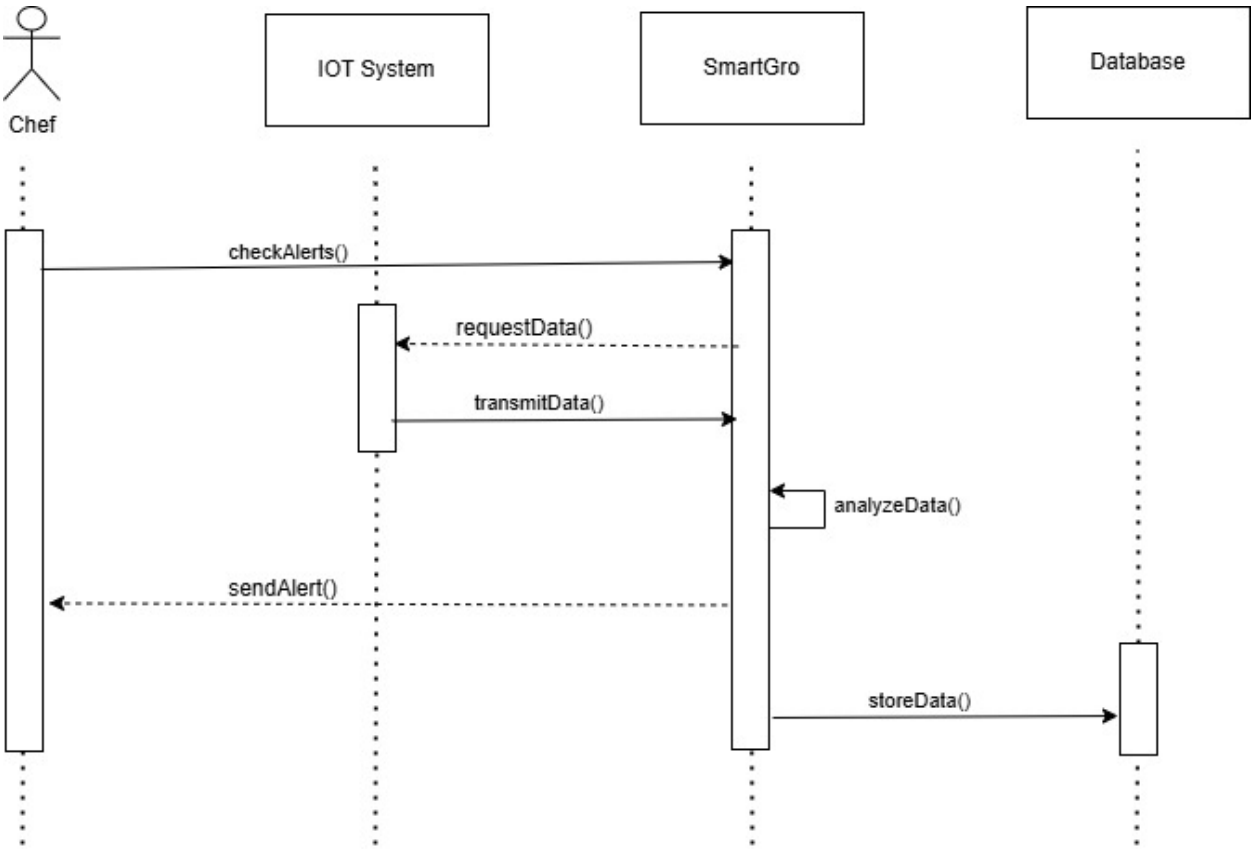
Class Diagram (without methods)



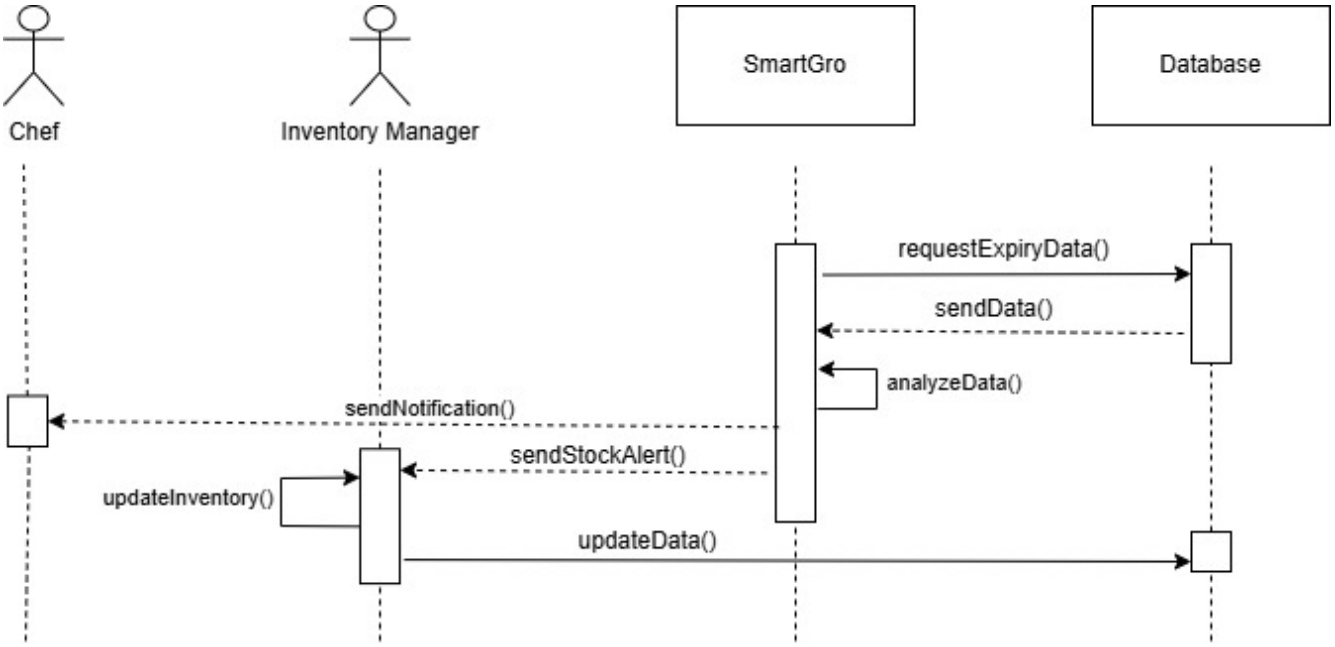
Object Behavior Model

Sequence Diagram

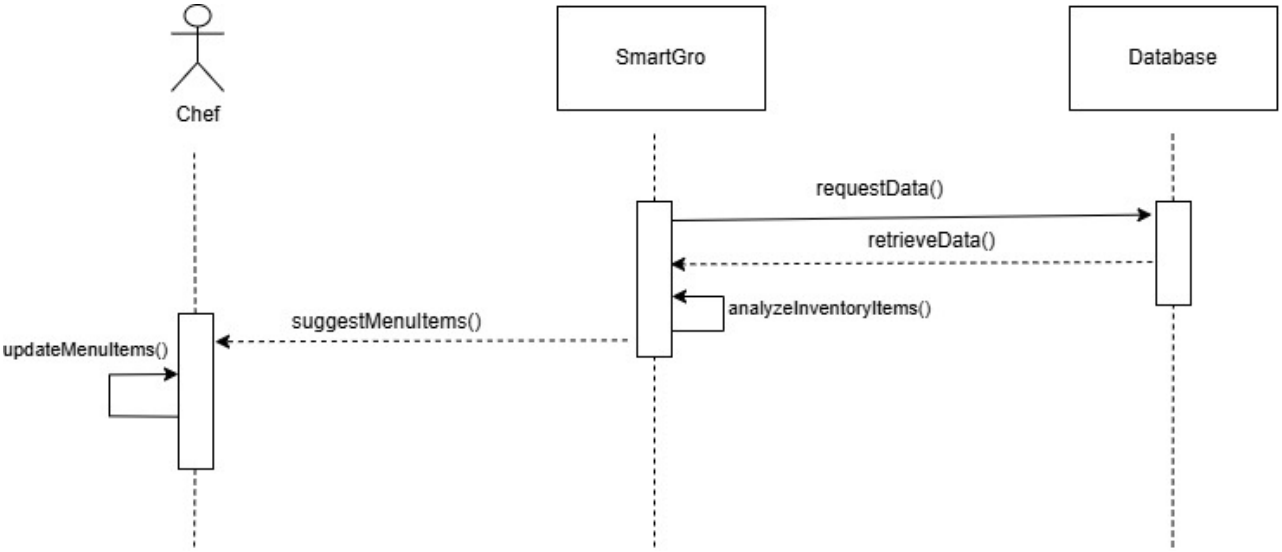
1. Real-Time Monitoring



2. Alert System Process



3. Menu Adjustment Process



Functional Specification

User Story 1: Real-Time Inventory Monitoring

- As an Inventory Manager, I want the IoT-enabled sensors to continuously track inventory conditions (quantity, temperature, humidity, and expiry dates) so that I can ensure optimal storage and prevent spoilage.

User Story 2: Automated Data Collection

- As an Inventory Manager, I want the system to automatically collect and process inventory data from IoT sensors so that I can reduce manual stock-taking efforts and improve accuracy.

User Story 3: Proactive Expiration Alerts

- As a Chef, I want to receive proactive alerts for items nearing expiry so that I can prioritize their usage and minimize food waste.

User Story 4: Menu Suggestions Based on Expiring Items

- As a Chef, I want the system to provide intelligent menu suggestions for expiring items so that I can create dishes and specials that make the best use of available ingredients.

User Story 5: Customizable Notification Settings

- As a Chef, I want to customize my notification preferences so that I can receive alerts during convenient times and avoid disruptions during peak hours.

User Story 6: Usage Pattern Analysis

- As an Inventory Manager, I want to analyze historical inventory usage patterns so that I can adjust stock levels accordingly and reduce excess or shortage.

User Story 7: Waste Trends and Reduction Recommendations

- As an Inventory Manager, I want the system to identify waste trends and recommend ordering adjustments so that I can minimize waste and improve efficiency.

User Story 8: Data-Driven Order Forecasting

- As an Inventory Manager, I want to receive stock forecasts based on past usage data so that I can optimize future orders and maintain the right inventory levels.

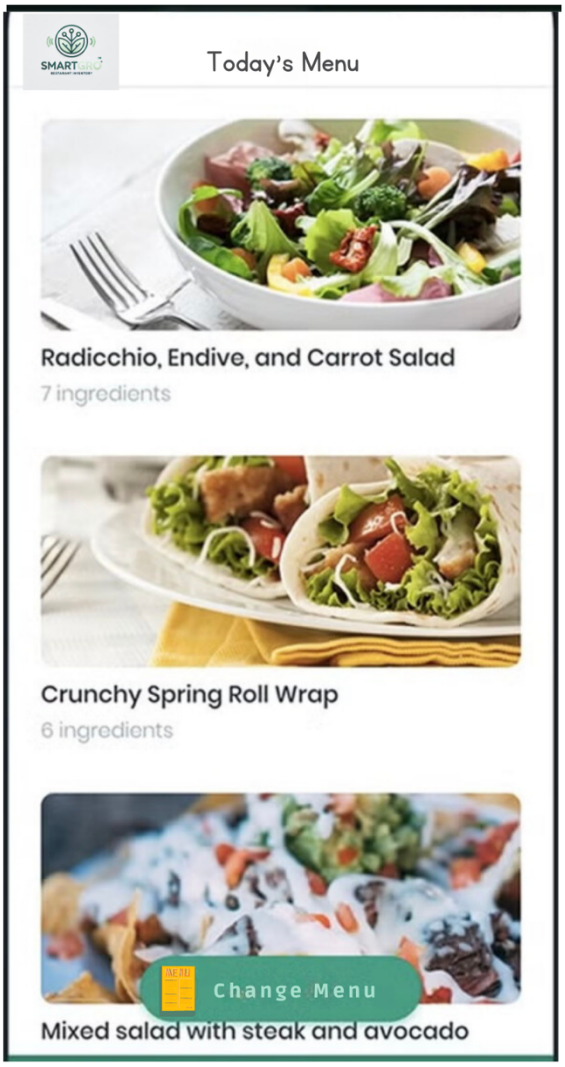
User Story 9: Automatic Reorder Triggers

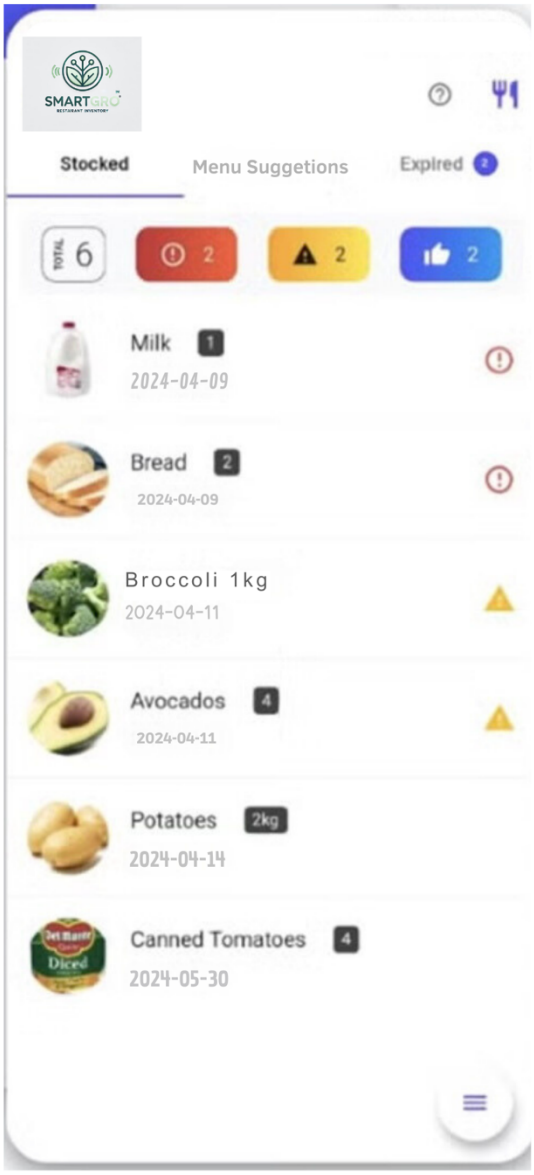
- As an Inventory Manager, I want the system to trigger reorder requests automatically when stock levels fall below thresholds so that I can prevent stockouts.

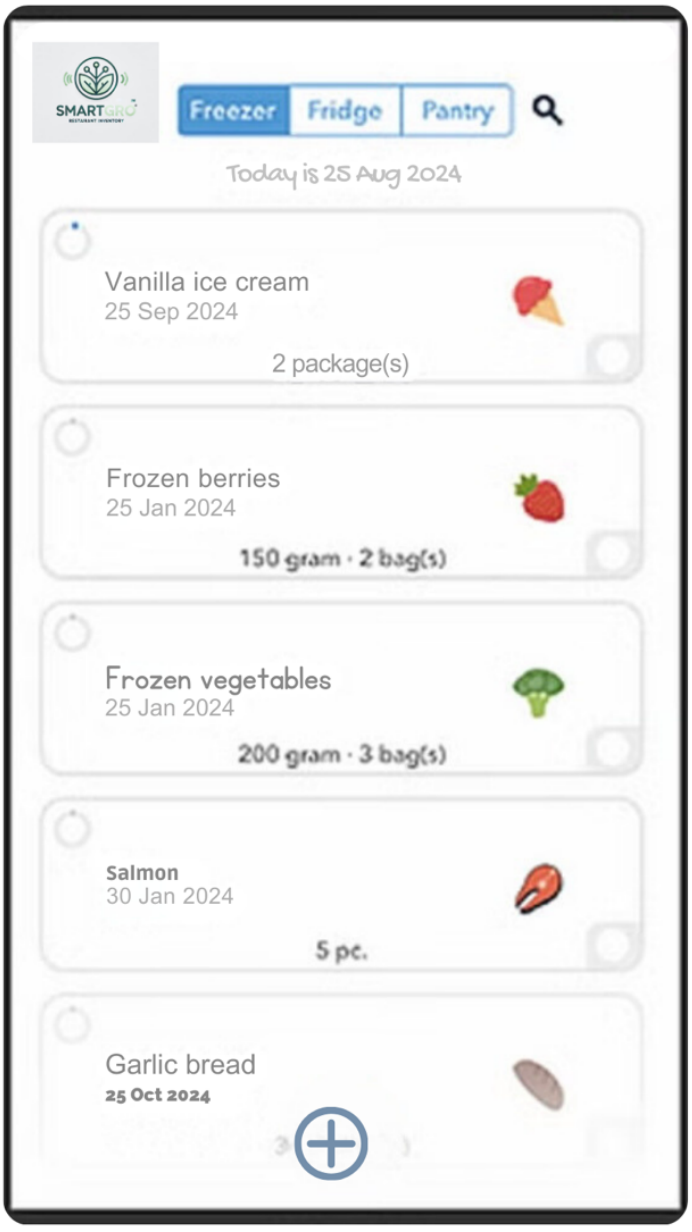
User Story 10: Supplier Integration and Order Management

- As an Inventory Manager, I want to manage orders directly through the system, including tracking deliveries and supplier updates, so that I can streamline the ordering process.

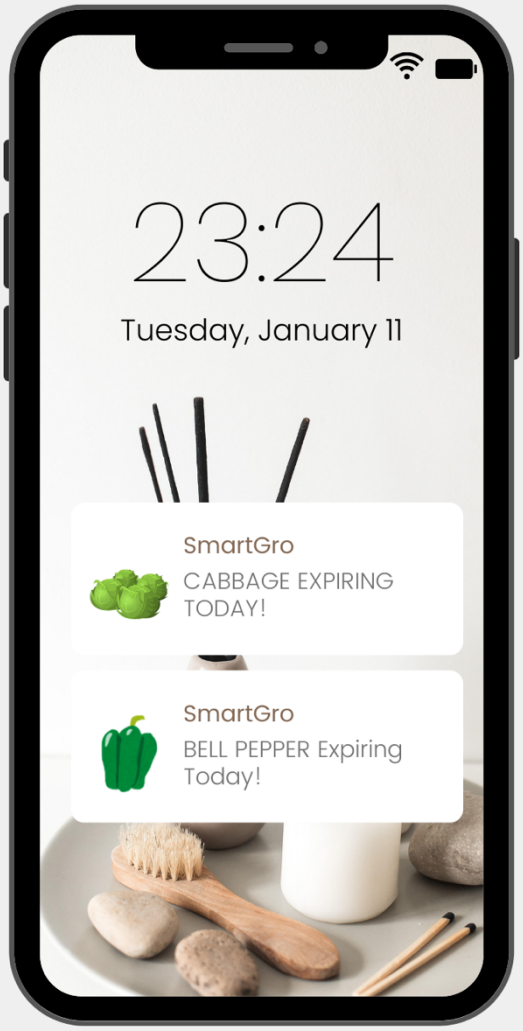
Interface Design



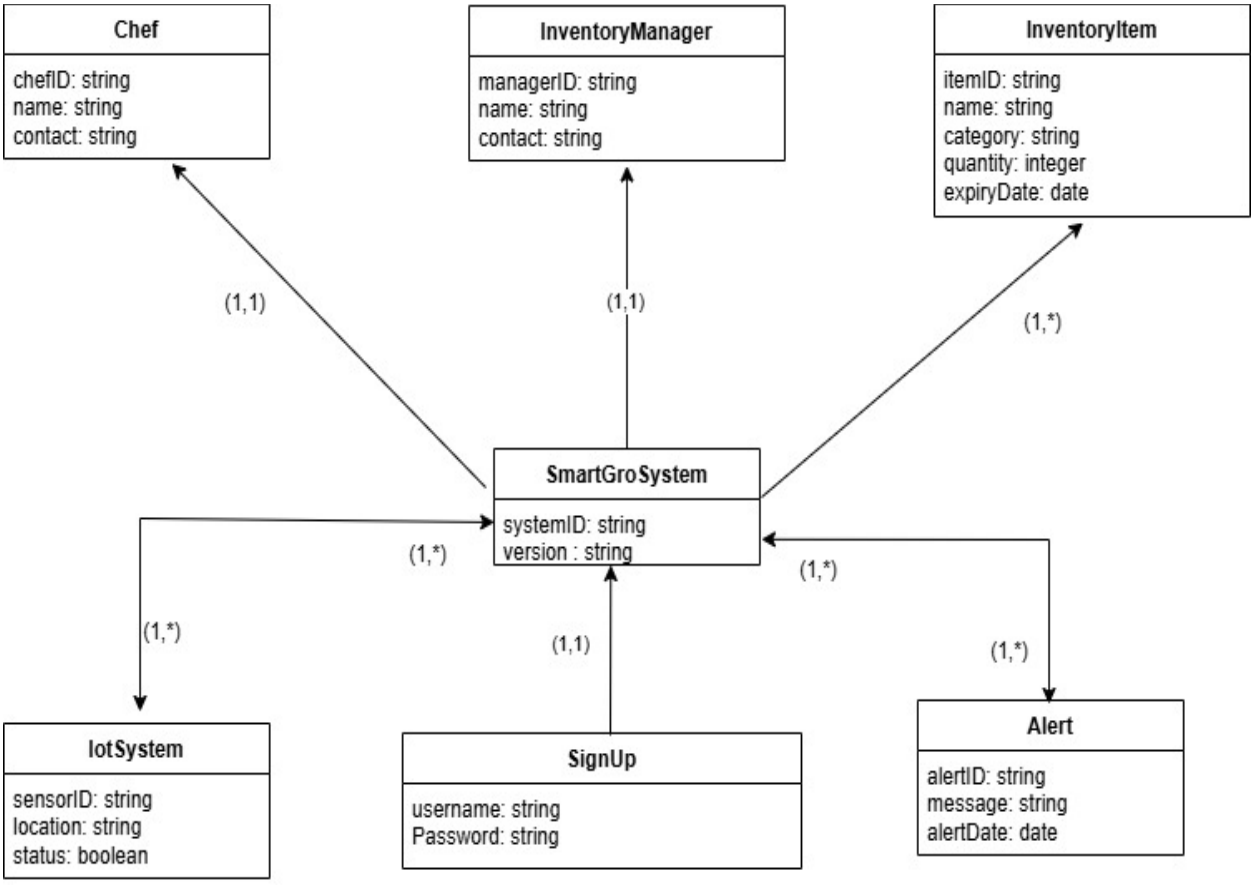

SMARTGRO
Stocked
Menu Suggetions
Expired
TOTAL 6
2
2
2
Milk 1
2024-04-09
Bread 2
2024-04-09
Broccoli 1kg
2024-04-11
Avocados 4
2024-04-11
Potatoes 2kg
2024-04-14
Canned Tomatoes 4
2024-05-30


SMARTGRO
Freezer
Fridge
Pantry
Today is 25 Aug 2024
Vanilla ice cream
25 Sep 2024
2 package(s)
Frozen berries
25 Jan 2024
150 gram · 2 bag(s)
Frozen vegetables
25 Jan 2024
200 gram · 3 bag(s)
Salmon
30 Jan 2024
5 pc.
Garlic bread
25 Oct 2024

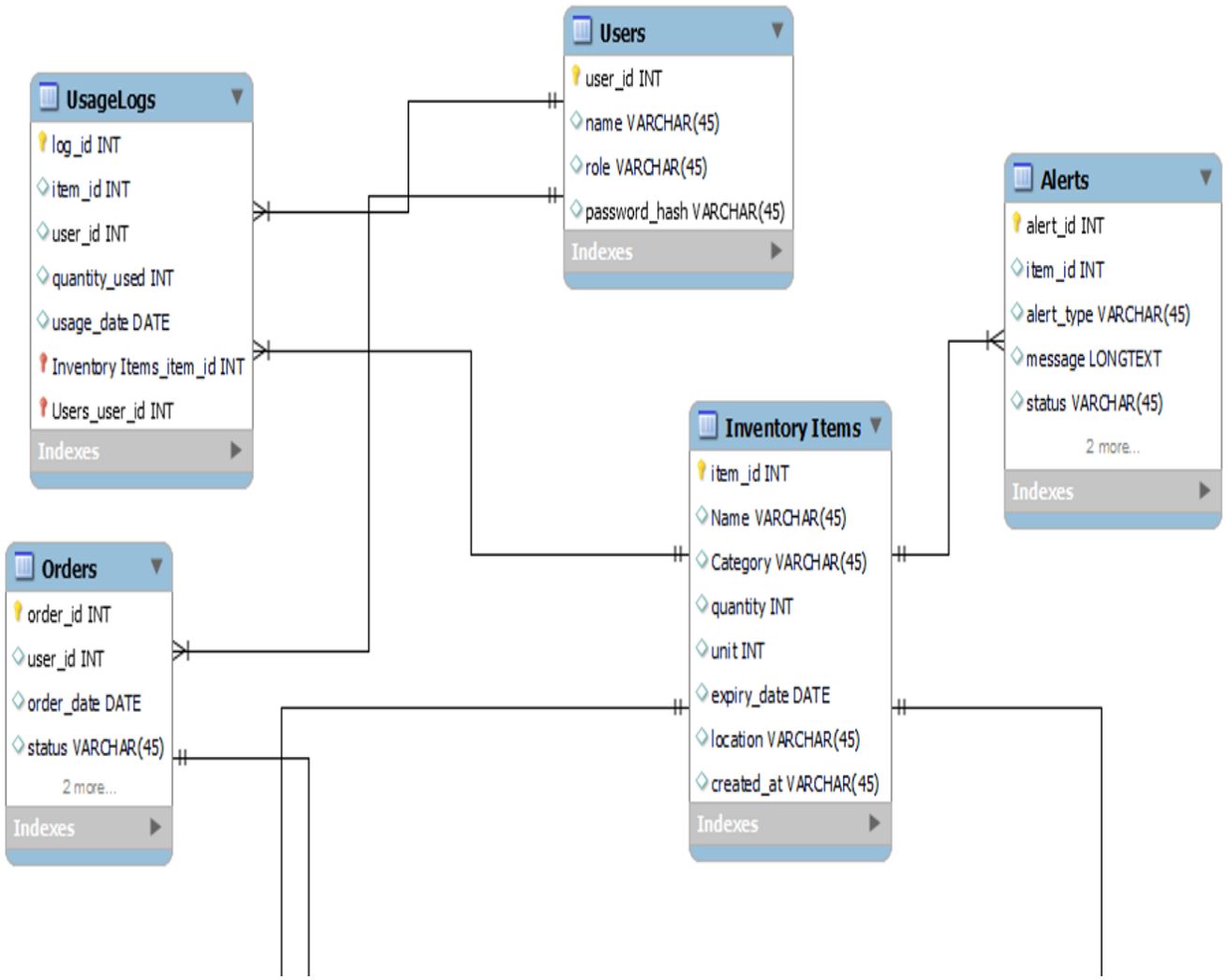
IF YOU DON'T REMEMBER YOUR TURN WE WILL NOTIFY YOU!

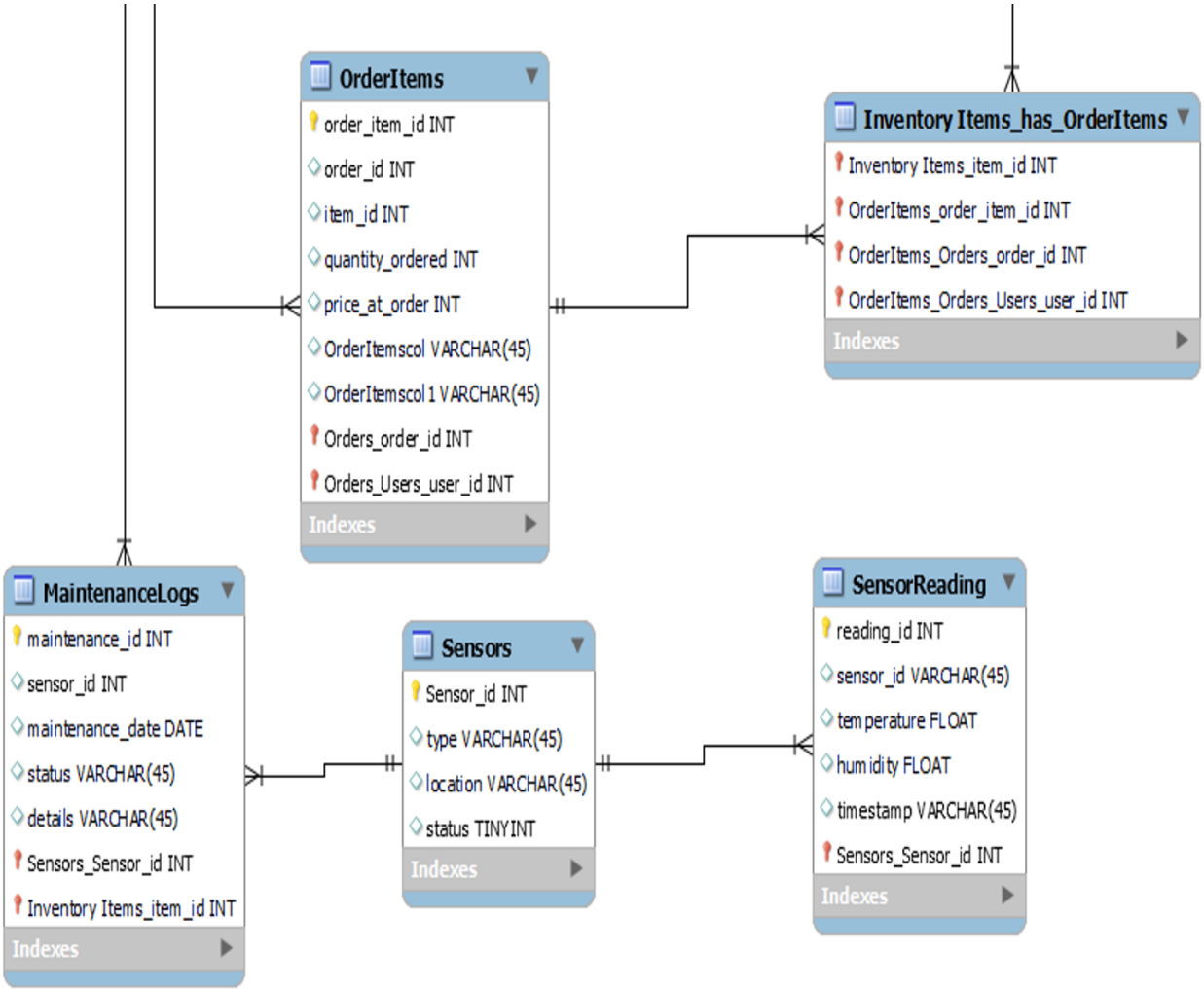


Class Diagram (with methods)



Database Design




OrderItems
order_item_id INT
order_id INT
item_id INT
quantity_ordered INT
price_at_order INT
OrderItemscol VARCHAR(45)
OrderItemscol1 VARCHAR(45)
Orders_order_id INT
Orders_Users_user_id INT
Indexes
Inventory Items_has_OrderItems
Inventory Items_item_id INT
OrderItems_order_item_id INT
OrderItems_Orders_order_id INT
OrderItems_Orders_Users_user_id INT
Indexes
MaintenanceLogs
maintenance_id INT
sensor_id INT
maintenance_date DATE
status VARCHAR(45)
details VARCHAR(45)
Sensors_Sensor_id INT
Inventory Items_item_id INT
Indexes
Sensors
Sensor_id INT
type VARCHAR(45)
location VARCHAR(45)
status TINYINT
Indexes
SensorReading
reading_id INT
sensor_id VARCHAR(45)
temperature FLOAT
humidity FLOAT
timestamp VARCHAR(45)
Sensors_Sensor_id INT
Indexes

Database Constraints

1. **InventoryItems Table**
- **Primary Key:** item_id
- **Not Null Constraints:** name, category, quantity, unit, expiry_date, location, created_at

2. **Sensors Table**
- **Primary Key:** sensor_id
- **Not Null Constraints:** type, location, status

3. **SensorReadings Table**
- **Primary Key:** reading_id
- **Foreign Key:** sensor_id references **Sensors(sensor_id)**
- **Not Null Constraints:** sensor_id, temperature, `humidity,timestamp`

4. **Users Table**
- **Primary Key:** user_id
- **Unique Constraint:** username
- **Not Null Constraints:** name, role, username, password_hash

5. **Alerts Table**
- **Primary Key:** alert_id
- **Foreign Key:** item_id references **InventoryItems(item_id)**
- **Not Null Constraints:** item_id, alert_type, message, status, created_at

6. **Orders Table**
- **Primary Key:** order_id
- **Foreign Key:** user_id references **Users(user_id)**
- **Not Null Constraints:** user_id, order_date, status, total_cost

7. **OrderItems Table**
- **Primary Key:** order_item_id
- **Foreign Keys:** order_id references **Orders(order_id),** item_id references **InventoryItems(item_id)**
- **Not Null Constraints:** order_id, item_id, quantity_ordered, price_at_order

8. **UsageLogs Table**

- **Primary Key:** log_id
- **Foreign Keys:** item_id references **InventoryItems(item_id),** user_id references **Users(user_id)**
- **Not Null Constraints:** item_id, user_id, quantity_used, usage_date

9. **MaintenanceLogs Table**

- **Primary Key:** maintenance_id
- **Foreign Key:** sensor_id references **Sensors(sensor_id)**
- **Not Null Constraints:** sensor_id, maintenance_date, status, details

Software Design

1. AddInventoryItem

Signature:

- Method Name: AddInventoryItem()
- Class Name: InventoryManagement
- ID: AddInventoryItemID
- Clients: Inventory Interface, Database

Responsibilities: Adds a new inventory item to the database with details such as name, category, quantity, and expiry date.
Arguments Received: Item Name, Category, Quantity, Unit, Expiry Date
Type of Value Returned: Status of Addition (Success or Failure)
Pre-Conditions: Ensure that the item does not already exist in the inventory under the same name and category.
Post-Conditions: Item details are stored in the Inventory database.
Logic:

1. IF no existing item with the same name and category:
    - VALIDATE entered item details.
    - IF details are valid:
        - CREATE a new inventory record.
        - STORE item details in InventoryItems table.
        - RETURN "Success"
    - ELSE
        - RETURN "Failure: Invalid Details"
2. ELSE
    - RETURN "Failure: Item Already Exists"

2. MonitorInventory

Signature:

- Method Name: MonitorInventory()
- Class Name: IoTSystem
- ID: MonitorInventoryID
- Clients: Sensors, Inventory Dashboard

Responsibilities: Continuously monitors inventory conditions (temperature, humidity) using IoT sensors.
Arguments Received: Sensor ID, Location
Type of Value Returned: Real-time Status (e.g., Normal, Alert)
Pre-Conditions: IoT sensors are calibrated and active.
Post-Conditions: Inventory status is updated in real-time on the dashboard.
Logic:

1. FOR EACH sensor:
    - COLLECT temperature and humidity data.
    - IF conditions deviate from safe ranges:
        - TRIGGER an alert.
        - UPDATE status in the Inventory Dashboard.
    - ELSE
        - RETURN "Normal"
2. END FOR

3. GenerateExpiryAlerts

Signature:

- Method Name: GenerateExpiryAlerts()
- Class Name: AlertSystem
- ID: GenerateExpiryAlertsID
- Clients: Chef Interface, Notification System

Responsibilities: Generates alerts for items nearing expiry and suggests menu options.
Arguments Received: Item ID, Expiry Date
Type of Value Returned: Alert Status (Sent or Not Sent)
Pre-Conditions: Inventory items must have expiry dates within the threshold period for alerts.
Post-Conditions: Chef is notified of expiring items, and menu suggestions are generated.
Logic:

1. FOR EACH item nearing expiry:
    - FETCH item details.
    - GENERATE alert message for the chef.
    - IF suggestions available:
        - ATTACH menu suggestions to alert.
        - SEND alert to chef’s interface.
        - RETURN "Sent"
    - ELSE
        - RETURN "Alert Generated Without Suggestions"
2. END FOR

4. PlaceReorder

Signature:

- Method Name: PlaceReorder()
- Class Name: OrderManagement
- ID: PlaceReorderID
- Clients: Inventory Manager, Supplier Integration

Responsibilities: Places reorder requests for low-stock items automatically.
Arguments Received: Item ID, Quantity Required, Supplier Details
Type of Value Returned: Reorder Status (Ordered or Error)
Pre-Conditions: Item quantity is below the reorder threshold.
Post-Conditions: Reorder request is sent to the supplier and recorded in the system.
Logic:

1. IF item quantity < reorder threshold:
    - FETCH supplier details for the item.
    - CREATE reorder request with specified quantity.
    - SEND reorder request to supplier.
    - RECORD reorder details in Orders table.
    - RETURN "Ordered"
2. ELSE
    - RETURN "Error: Quantity Sufficient

5. ProcessSensorData

Signature:

- Method Name: ProcessSensorData()
- Class Name: SensorProcessing
- ID: ProcessSensorDataID
- Clients: IoT System, Inventory Analytics

Responsibilities: Processes data from IoT sensors and logs readings in the database.
Arguments Received: Sensor ID, Temperature, Humidity, Timestamp
Type of Value Returned: Processing Status (Success or Error)
Pre-Conditions: Sensor readings are transmitted to the system in real time.
Post-Conditions: Sensor readings are recorded in the SensorReadings table for analysis.
Logic:

1. RECEIVE temperature, humidity, and timestamp from sensors.
2. VALIDATE data format and completeness.
3. IF data is valid:
    - STORE reading in SensorReadings table.
    - RETURN "Success"
4. ELSE
    - RETURN "Error: Invalid Data"

6. CalculateUsageTrends

Signature:

- Method Name: CalculateUsageTrends()
- Class Name: AnalyticsEngine
- ID: CalculateUsageTrendsID
- Clients: Inventory Manager, Analytics Dashboard

Responsibilities: Analyzes usage patterns based on historical data to forecast future needs.
Arguments Received: Item ID, Time Period
Type of Value Returned: Usage Trend Data
Pre-Conditions: Valid time period and item data in the system.
Post-Conditions: Usage trend analysis is updated in the analytics dashboard.
Logic:

1. FETCH historical usage data for the specified item and period.
2. ANALYZE usage trends over the specified period.
3. CALCULATE average usage rate.
4. FORECAST future needs based on calculated usage rate.
5. RETURN trend data.

7. UpdateInventory

Signature:

- Method Name: UpdateInventory()
- Class Name: InventoryManagement
- ID: UpdateInventoryID
- Clients: Chef, Inventory Manager

Responsibilities: Updates inventory records after items are used or replenished.
Arguments Received: Item ID, New Quantity
Type of Value Returned: Update Status (Success or Failure)
Pre-Conditions: Valid item ID and positive quantity.
Post-Conditions: Inventory record is updated in the database.
Logic:

1. LOCATE item by Item ID in InventoryItems table.
2. IF item exists:
    - UPDATE quantity with new value.
    - RETURN "Success"
3. ELSE
    - RETURN "Failure: Item Not Found"

Executive Summary

SmartGro is a revolutionary tool in the restaurant industry, designed to streamline inventory management, reduce food waste, and optimize menu planning through intelligent IoT-based solutions. The system addresses critical challenges faced by restaurants, including the complexities of tracking perishable inventory, the risk of spoilage, and the inefficiencies in manual stock-taking. With SmartGro, restaurants gain access to a powerful platform that integrates IoT technology, predictive analytics, and real-time alerts to enhance operational efficiency and promote sustainability.

SmartGro's **IoT-Enabled Inventory Monitoring (IIM)** feature empowers restaurants to maintain optimal storage conditions and keep a real-time, accurate view of inventory. By utilizing sensors to monitor temperature, humidity, and stock levels, SmartGro ensures that items are stored under ideal conditions, thus preventing spoilage and waste. This real-time tracking reduces the reliance on manual inventory processes and provides a reliable foundation for inventory decisions.

To further minimize waste and maximize kitchen creativity, SmartGro introduces the **Smart Expiry Alert and Menu Suggestion System (SEAMSS)**. SEAMSS proactively notifies chefs about items nearing expiry and offers recipe suggestions to make the most of these ingredients. This system not only helps reduce food waste but also allows restaurants to create new dishes and specials that utilize ingredients efficiently.

The **Dynamic Inventory Analytics (DIA)** feature is a game-changer for inventory managers. By analyzing historical usage data and identifying consumption patterns, DIA offers actionable insights into inventory trends, waste reduction strategies, and optimized ordering cycles. This enables managers to make data-driven decisions that improve inventory turnover, reduce excess stock, and ensure that popular items are readily available.

SmartGro's **Automated Stock Replenishment System (ASRS)** streamlines the reordering process by automatically generating order requests when stock levels fall below predefined thresholds. ASRS not only integrates seamlessly with suppliers but also helps managers track deliveries and manage inventory more effectively, eliminating the risk of stockouts and ensuring essential items are always on hand.

Additionally, **Predictive Maintenance for IoT Sensors (PMIS)** ensures the longevity and reliability of IoT devices by sending proactive maintenance alerts. By monitoring sensor performance, PMIS reduces the chances of system downtime, providing uninterrupted monitoring and optimal performance.

With its features of IIM, SEAMSS, DIA, ASRS, and PMIS, SmartGro offers a unique, comprehensive solution that transforms inventory management in the restaurant industry. By boosting operational efficiency, reducing food waste, and enhancing menu planning, SmartGro not only meets the current demands of the restaurant sector but also paves the way for a more sustainable and technology-driven future in food service.

Project Presentation Link

https://www.canva.com/design/DAGWfDMQDBQ/gsQZAIzglmDYQcsP-8fGKA/edit?utm_content=DAGWfDMQDBQ&utm_campaign=designshare&utm_medium=link2&utm_source=sharebutton

Project Recording Link

Microsoft Teams meeting-20241201_155537-Meeting Recording.mp4

Project Management Deliverables

Weekly Project Timeline

Planned Due Date	Actual Completion Date	Tasks
Sep 6, 2024	Sep 6, 2024	Executive Summary
Sep 6, 2024	Sep 6, 2024	Problem Statement
Sep 15, 2024	Sep 15, 2024	Business Process Model
Sep 22, 2024	Sep 22, 2024	Context Diagram
Sep 29, 2024	Sep 29, 2024	Use Case Diagram
Sep 29, 2024	Sep 29, 2024	Use Case Descriptions
Oct 6, 2024	Oct 6, 2024	Data Dictionary
Oct 13, 2024	Oct 13, 2024	Class Diagram without Methods
Oct 13, 2024	Oct 13, 2024	Sequence Diagram
Oct 20, 2024	Oct 20, 2024	Functional Specification Document
Oct 20, 2024	Oct 20, 2024	Interface Design

Planned Due Date	Actual Completion Date	Tasks
Oct 20, 2024	Oct 20, 2024	Class Diagram with Methods
Oct 27, 2024	Oct 27, 2024	Database Design
Oct 27, 2024	Oct 27, 2024	Software Design
Nov 3, 2024	Nov 3, 2024	Presentation
Nov 3, 2024	Nov 3, 2024	Project Report Due

Project Minutes

Meeting #1

- **Date and Time**: Sep 6, 2024, at 3:00 PM–4:00 PM
- **Attendees**: Radhika, Sai Bhavana, Aditya
- **Discussion**: Completed executive summary and problem statement tasks after the introduction to system concepts (Aug 26–Sep 1).

Meeting #2

- **Date and Time**: Sep 15, 2024, at 2:00 PM–3:30 PM
- **Attendees**: Radhika, Sai Bhavana, Aditya
- **Discussion**: Finalized the business process model after learning about system proposals and requirements collection (Sep 9–15).

Meeting #3

- **Date and Time**: Sep 22, 2024, at 10:00 AM–12:00 PM
- **Attendees**: Radhika, Sai Bhavana, Aditya
- **Discussion**: Completed the context diagram after covering system modeling (Sep 16–22).

Meeting #4

- **Date and Time**: Sep 29, 2024, at 2:30 PM–4:00 PM
- **Attendees**: Radhika, Sai Bhavana, Aditya
- **Discussion**: Developed case diagrams and case descriptions after process modeling and user stories (Sep 23–29).

Meeting #5

- **Date and Time**: Oct 6, 2024, at 3:00 PM–5:00 PM
- **Attendees**: Radhika, Sai Bhavana, Aditya

- **Discussion**: Finalized the data dictionary after completing data modeling (Sep 30–Oct 6).

Meeting #6

- **Date and Time**: Oct 13, 2024, at 2:00 PM–4:00 PM
- **Attendees**: Radhika, Sai Bhavana, Aditya
- **Discussion**: Reviewed class diagram without methods and sequence diagrams following dynamic modeling (Oct 7–13).

Meeting #7

- **Date and Time**: Oct 20, 2024, at 9:30 AM–11:00 AM
- **Attendees**: Radhika, Sai Bhavana, Aditya
- **Discussion**: Completed functional specification document, interface design, and class diagram with methods after UML model analysis (Oct 14–20).

Meeting #8

- **Date and Time**: Oct 27, 2024, at 1:00 PM–3:00 PM
- **Attendees**: Radhika, Sai Bhavana, Aditya
- **Discussion**: Finalized database design and software design tasks based on system design topics (Oct 21–27).

Meeting #9

- **Date and Time**: Nov 3, 2024, at 3:00 PM–5:00 PM
- **Attendees**: Radhika, Sai Bhavana, Aditya
- **Discussion**: Prepared and submitted the presentation and project report after covering IS project management topics (Oct 29–Nov 3).

References

https://www.intuz.com/blog/applications-of-iot-in-inventory-management

https://restauranttechnologynews.com/2023/09/how-the-internet-of-things-iot-can-help-restaurants-maximize-growth-and-streamline-operations/